

www.Info Support.com



Ontwerp en analyse document

Afstudeeropdracht: Paaspop de ultieme festival App

Merik Westerveld - 2737450

15 februari 2019

Concept

**Hoofdkantoor**

Kruisboog 42

3905 TG Veenendaal

Nederland

Tel. +31(0)318 - 55 20 20

Fax +31(0)318 - 55 23 55

**Kenniscentrum**

De Smalle Zijde 39

3903 LM Veenendaal

Tel. +31(0)318 - 50 11 19

Fax +31(0)318 - 51 83 59

info.nl@Info Support.com

www.Info Support.com

K.v.K 3013 5370

BTW NL8062.30.277B01

IBAN NL92 RABO 0305 9528 89

BIC RABONL2U

IBAN NL74 INGB 0004 7385 93

BIC INGBNL2A

Ontwerp en analyse document

Afstudeeropdracht: Paaspop de ultieme festival App

|  |  |
| --- | --- |
| Gegevens |  |
|  | Ontwerp en analyse document |
|  | Afstudeeropdracht: Paaspop de ultieme festival App |
|  | 1.0 |
|  | Concept |
|  | 15 februari 2019 |
|  | Ontwerp en analyste document – Merik Westerveld – Info Support |
|  | Info Support |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1.0 | Concept | 15 Februari 2019 | Merik Westerveld | Opzet document |
| 1.1 | Concept | 22 Februari 2019 | Merik Westerveld | Feedback verwerkt op de functionele en niet functionele eisen en het ontwerp |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1.0 | Concept | 18 Februari 2019 | Hans Geurtsen |

**© Info Support B.V., Veenendaal 2019**

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze ook, zonder voorafgaande toestemming van **Info Support B.V.**

No part of this publication may be reproduced in any form by print, photo print, microfilm or any other means without written permission by **Info Support B.V.**

Prijsopgaven en leveringen geschieden volgens de Algemene Voorwaarden van **Info Support B.V.** gedeponeerd bij de K.v.K. te Utrecht onder nr. 30135370. Een exemplaar zenden wij u op uw verzoek per omgaande kosteloos toe.

# **Inleiding**

# **Inhoudsopgave**

[1. Inleiding 3](#_Toc1394052)

[Inhoudsopgave 4](#_Toc1394053)

[2. Eisen 5](#_Toc1394054)

[2.1 Functionele eisen 5](#_Toc1394055)

[2.1.1 Mobiele applicatie 5](#_Toc1394056)

[2.1.2 Dashboard 6](#_Toc1394057)

[2.2 Niet-functionele eisen 6](#_Toc1394058)

[3. Mockup mobiele app 7](#_Toc1394059)

# Eisen

## Functionele eisen

### Mobiele applicatie

|  |  |  |
| --- | --- | --- |
| ID | Omschrijving | MoSCoW |
| FEM1. | De gebruiker moet bij de eerste keer gebruik van de app aan kunnen geven of de festivalbezoeker een man / vrouw / gender neutraal is en de leeftijd. | M |
| FEM2. | De gebruiker moet in kunnen zien welke artiest waar en wanneer spelen. | M |
| FEM3. | De gebruiker moet kunnen zien welke bar en wc het meest geschikt is om naar toe te gaan wat wordt gebaseerd op afstand en drukte. | M |
| FEM4. | De gebruiker moet een plattegrond van het festival kunnen zien. | M |
| FEM5. | De gebruiker moet aan kunnen geven naar welke artiest hij / zij wilt. | M |
| FEM6. | De gebruiker moet de weersverwachting kunnen zien van het festival. | S |
| FEM7. | De gebruiker moet om een bepaalde tijd een melding krijgen dat die water moet drinken. | S |
| FEM8. | De gebruiker moet een meeting punt kunnen laten genereren voor de beste plek om zijn / haar vrienden te ontmoeten. | S |
| FEM9. | De gebruiker moet zijn / haar gegevens kunnen verwijderen, zowel lokaal als wat in de cloud staat opgeslagen. | S |
| FEM10. | De gebruiker moet het interesse level van een artiest in kunnen zien. | S |
| FEM11. | De gebruiker moet een push melding krijgen 10 minuten voor aanvang van een artiest. | S |
| FEM12. | De gebruiker moet op basis van zijn / haar interesse aanraders krijgen. | C |
| FEM13. | De gebruiker moet zijn budget aan kunnen geven en bijhouden hoeveel muntjes / geld hij of zij nog over heeft. | C |
| FEM14. | De gebruiker kan een paklijst aanmaken en aanvinken wat hij / zij heeft. | W |
| FEM15. | De gebruiker kan zijn / haar tent op een plaats neerzetten welke aangeeft waar die staat. | W |
| FEM16. | De gebruiker moet de app kunnen laten kiezen naar welke artiest hij / zij wilt. | W |

### Dashboard

|  |  |  |
| --- | --- | --- |
| ID | Omschrijving | MoSCoW |
| FED1. | De gebruiker moet aan kunnen geven welke artiest waar en wanneer speelt. | C |
| FED2. | De gebruiker moet in kunnen zien welke hoe populair elke artiest is. | C |
| FED3. | De gebruiker moet een push melding kunnen sturen naar alle gebruikers van de mobiele applicatie. | C |

## Niet-functionele eisen

Deze niet-functionele eisen zijn gebaseerd op ISO 9126.

|  |  |  |
| --- | --- | --- |
| ID | Omschrijving | MoSCoW |
| NFEM1. | De app moet zonder internet ook beschikbaar zijn (niet alle functionaliteiten). | M |
| NFEM2. | Ongeautoriseerde personen mogen niet in het dashboard komen. | M |
| NFEM3. | De app en het opslaan van data moet voldoen aan de privacy wetgeving. | M |
| NFEM4. | Voor beide de app en het dashboard moet de software goed gedocumenteerd zijn zodat op deze manier de kennis kan worden overdragen. | M |
| NFEM5. | Alle software moet op een juiste manier getest worden (unit testen, integratie testen en statische code kwaliteit testen). | M |
| NFEM6. | De app moet voldoen aan de huisstijl van Paaspop. | M |
| NFEM7. | De mobiele app moet beschikbaar zijn op zowel IOS als Android. | M |
| NFEM8. | Beide de app en het dashboard mogen geen laadtijden hebben die langer dan 3 seconden duren. | S |
| NFEM9. | De app moet zonder locatietoegang ook werken (niet alle functionaliteiten. | S |
| NFEM10. | De app moet getest zijn door middel van UX testen. | S |
| NFEM11. | De app moet op beide platformen native uitzien (design). | C |

# Mockup mobiele app

1: Scherm één is het ‘splash screen’, ook wel het eerste scherm wat de gebruiker te zien krijgt waar ondertussen de app laadt. Hier wordt het Paaspop logo op afgebeeld zodat de gebruiker gelijk kennis maakt met de huisstijl die verder in de app gebruikt wordt.

2: Dit tweede scherm komt de gebruiker van de app maar één keer in zijn gebruik tegen, tenzij de gebruiker er voor kiest om alle data te verwijderen (mogelijk in scherm 6) en de app daarna weer opnieuw te gebruiken. Op dit scherm kies de gebruiker de sekse waar de persoon zich mee identificeert. Ook bepalen ze de leeftijd. Na het invullen van deze data kan verder gegaan worden met het gebruiken van de app.

3: Het derde scherm is het scherm waar de gebruiker op terecht komt wanneer de gebruiker scherm twee al minimaal één keer doorlopen heeft. Op dit scherm kan de gebruiker de plattegrond zien en zijn / haar locatie. Daarnaast kan er op de knop ‘Meeting punt’ gedrukt worden waardoor de app een meeting punt kan genereren welke de gebruiker naar zijn / haar vrienden kan sturen.

4: Op scherm vier kan de gebruiker per stage zien welke artiest er wanneer staat. Hier kan die een artiest een ‘hartje’ geven waardoor die in de lijst komt te staan van artiesten die de gebruiker wilt zien op het festival. Ook kan er op een artiest geklikt worden waarna de gebruiker op scherm 7 komt.

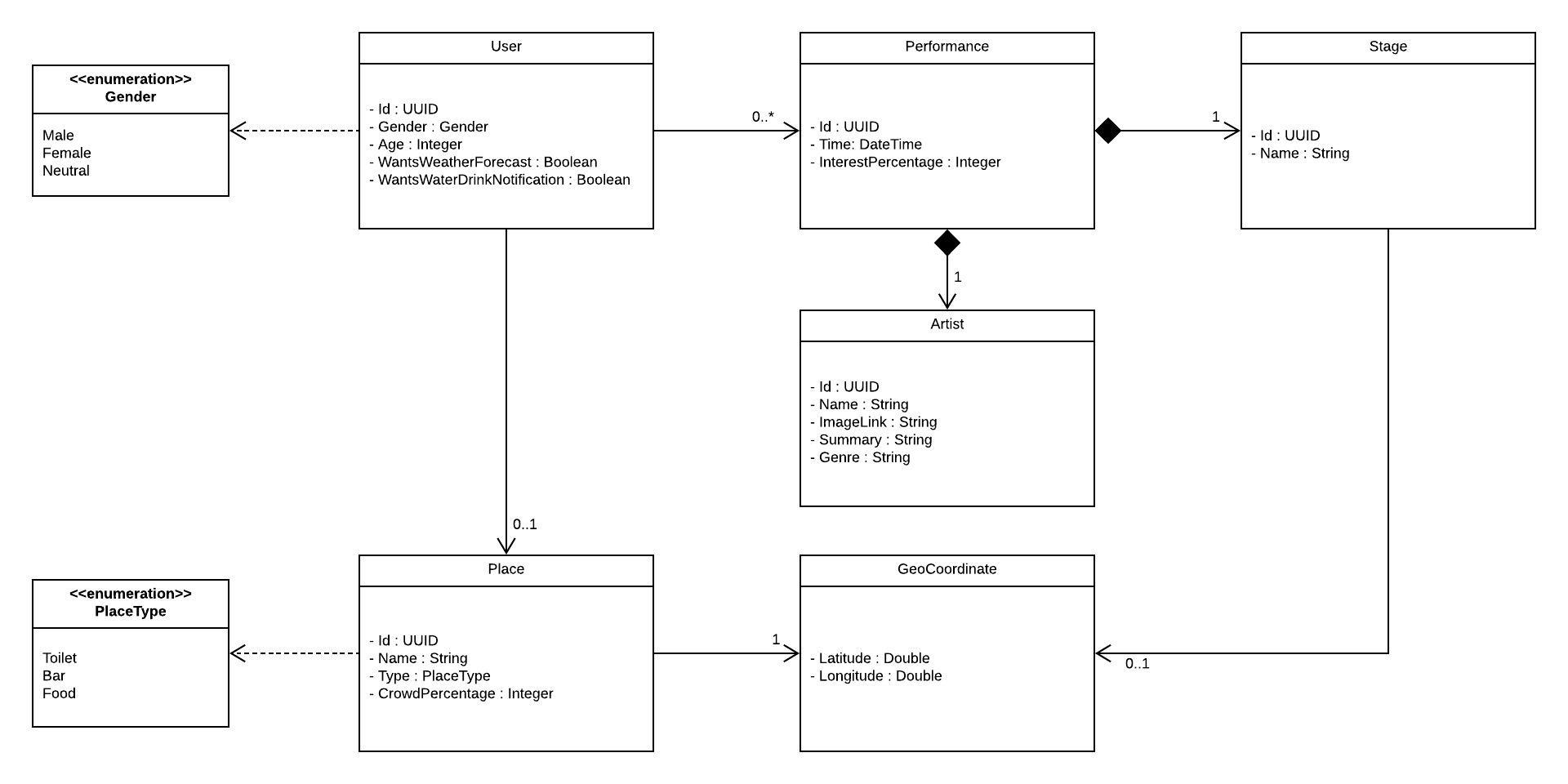
5: Scherm 5 wordt gebruikt als ‘plekken’ scherm waar de gebruiker kan zien wat de beste wc, bar of eettent is op basis van de afstand van de gebruiker en de drukte van de locatie. Als de gebruiker op een plek klikt gaat deze door naar scherm 8.

6: Het laatste scherm in de tab navigator is het ‘zelf’ scherm. Hier kan de gebruiker kiezen of hij / zij de weersverwachting of water drink melding wilt krijgen. Ook kan de gebruiker hier ervoor kiezen om zijn account te verwijderen en kan hij / zij zijn / haar rooster van favoriete artiesten inzien.

7: Via scherm 4 kan de gebruiker naar scherm 7 gaan waar hij / zij meer informatie krijgt te zien over de artiest en staat er een indicator hoeveel interesse er is voor deze artiest.

8: Via scherm 5 komt de gebruiker bij scherm 8 waar het locatie punt getoond wordt van de plek die is aangeklikt en de locatie van de gebruiker zodat deze er makkelijk naartoe kan.

# ERD (Entity-relationshipmodel)



User: De gebruiker heeft een uniek ID waaraan ook de app gekoppeld wordt. Daarnaast wordt er bijgehouden met welk geslacht de gebruiker zich identificeert en de leeftijd. Of de gebruiker een weersverwachting en of water drink melding wil wordt hier ook bijgehouden. De plek wordt ook bijgehouden bij de gebruiker om bij te houden waar de gebruiker is en een lijst met optredens.

Performance: Performance is een combinatie van de stage en de artiest. In deze klasse wordt ook de tijd / dag bijgehouden van het optreden. Een performance kan alleen bestaan met een stage en artiest. Als laatste wordt er een interesse percentage bijgehouden hoeveel app gebruikers naar dit optreden willen.

Artist: De artiest heeft een naam, imagelink waardoor het plaatje getoond kan worden in de app, een kleine beschrijving van de artiest en het genre van muziek. Deze klasse is gelinkt aan het optreden.

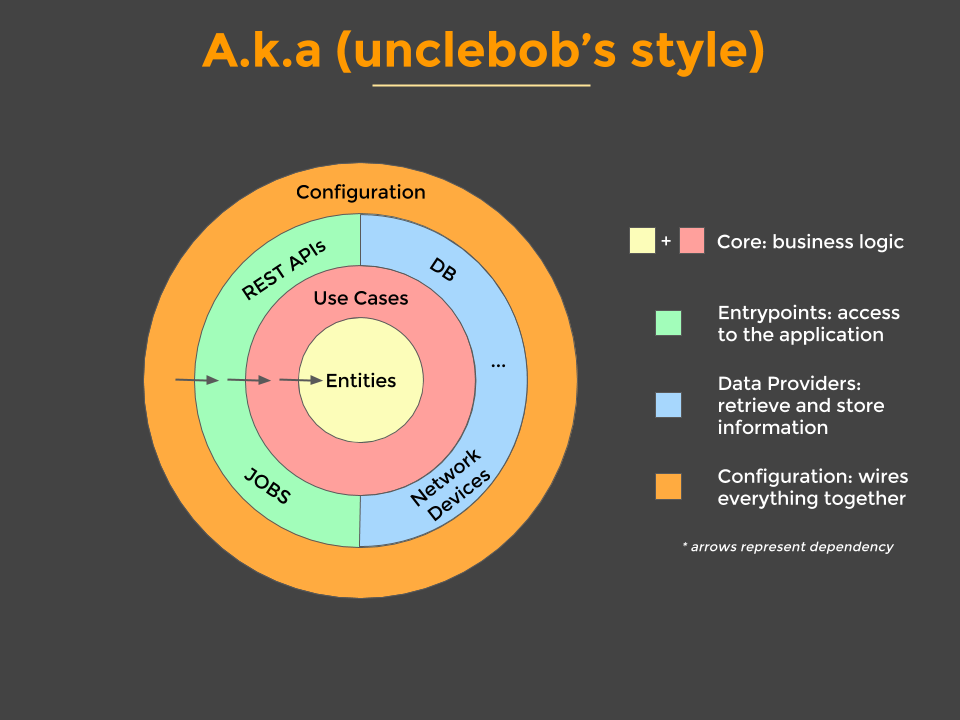
Stage: De stage hoort net als de artiest bij de performance. Verder heeft de stage een naam en optioneel een locatie.

Place: Een plek is een toilet, bar of eettent. Hieraan zit een naam verbonden en een crowd percentage om bij te houden hoe druk een bepaalde plek is. Daarnaast heeft deze klasse net als een stage een locatie. Echter kan deze locatie niet optioneel zijn.

GeoCoordinate: Een locatie object waar nu de latitude en longitude wordt bijgehouden.

# Architectuur backend

## Clean architecture

De backend architectuur is gebaseerd op clean architecture, een ontwerp bedacht door Robert Cecil Martin ook wel bekend als uncle bob. Deze architectuur is gebaseerd op de volgende vijf elementen [[1]](#footnote-1):

1. Onafhankelijk van frameworks.

2. Alle business rules moeten kunnen worden getest zonder de UI, database en alle andere externe elementen.

3. Onafhankelijk van de UI.

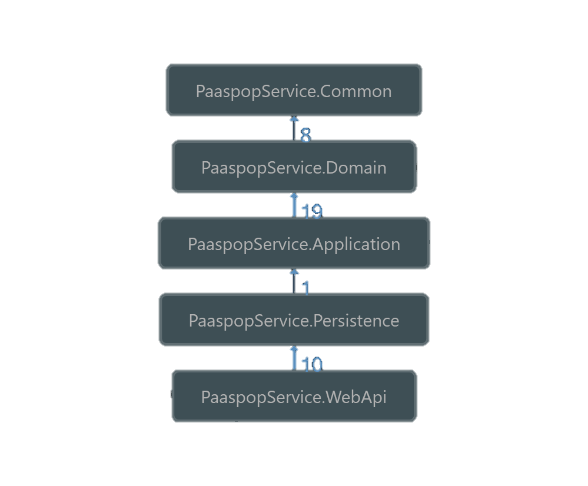
4. Onafhankelijk van de Database.

5. Onafhankelijk van enige andere externe partijen.

In de afbeelding hieronder staat de architectuur beschreven in een diagram.

Figuur 1: diagram van clean architecture [[2]](#footnote-2)

De code is opgedeeld in verschillende lagen welke allemaal hun eigen verantwoordelijkheid hebben. In de backend die voor Paaspop wordt gebruikt ziet het diagram er als volgt uit:



PaaspopService.WebApi: is de laag die verantwoordelijk is   
 Voor de UI / API. Di tis ook wel het entrypoint.

PaaspopService.Persistence: de laag die verbinding maakt met de desbetreffende database.

PaaspopService.Application: In deze laag worden de use cases beschreven, bijvoorbeeld een gebruiker vinden.

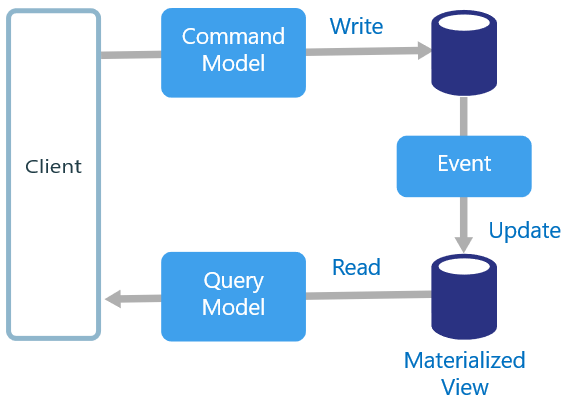
PaaspopService.Domain: de entiteiten laag (vormt samen PaaspopService.Application de core van de applicatie).

PaaspopService.Common: elementen van de applicatie die ook zonder aanpassingen in andere applicaties gebruikt zou kunnen worden.

Er is voor deze architectuur gekozen omdat het onnodige koppelingen weg haalt in een applicatie waardoor de applicatie beter test- en uitwisselbaar is.

## CQRS

CQRS, ook wel Command and Query Responsibility Segregation, is een patroon waarbij de verantwoordelijkheid tussen het schrijven (command) en het lezen (queries) van data is gescheiden.



Er is voor dit patroon gekozen om de modellen van het schrijven en lezen van elkaar te scheiden waardoor er wordt gegarandeerd dat enkel de juiste modellen de juiste acties uitvoeren. Daarnaast is het makkelijker schaalbaar (eventueel met het creëren van een aparte schrijf en aparte lees database), onderhoudbaar (duidelijke scheiding waar wat geïmplementeerd moet worden) en uitwisselbaar (mochten de lees queries een andere koppeling nodig hebben dan de schrijf commands). [[3]](#footnote-3)

## Third party libraries

### Mediatr

Mediatr is een third party library waarmee binnen de applicatie een pipeline opgezet kan worden waar een request doorheen gaat. Dit werkt goed samen met CQRS (5.2) en FluentValidation (5.3.3). Daarnaast worden de request objecten en handlers gescheiden van elkaar. Hieronder staat een beschrijving van een request:

public class GetArtistQuery : IRequest<ArtistViewModel>

{

public string Id { get; set; }

}

Vervolgens is er een handler, welke verbinding maakt met een database implementatie, die gebruik maakt van het hierboven beschreven request:

public class GetArtistsQueryHandler : GeneralRequestHandler<GetArtistQuery, ArtistViewModel>

{

private readonly IArtistsRepository \_artistsRepository;

public GetArtistsQueryHandler(IMapper mapper, IArtistsRepository artistsRepository)

: base(mapper)

{

\_artistsRepository = artistsRepository;

}

public override async Task<ArtistViewModel> Handle(GetArtistQuery request, CancellationToken cancellationToken)

{

Artist result = await \_artistsRepository.GetArtistById(request.Id, cancellationToken);

return Mapper.Map<ArtistViewModel>(result);

}

}

Deze handler wordt door de mediatr library en dependency injection via de controller uitgevoerd:

[Route("api/[controller]")]

[ApiController]

public class ArtistsController : BaseController

{

[HttpGet("{id}", Name = "Get")]

public async Task<ActionResult<ArtistViewModel>> Get(string id)

{

return Ok(await GetMediator().Send(new GetArtistQuery {Id = id}));

}

}

Al deze requests gaan door een pipeline wat elementen bevat zoals een logger. Deze logger is een IRequestPreProcessor wat een voorbeeld is van een pipeline element wat vóór een request dankzij het gebruik van mediatr wordt uitgevoerd[[4]](#footnote-4):

public class RequestLogger<TRequest> : IRequestPreProcessor<TRequest>

{

private readonly ILogger \_logger;

public RequestLogger(ILogger<TRequest> logger)

{

\_logger = logger;

}

public Task Process(TRequest request, CancellationToken cancellationToken)

{

var name = typeof(TRequest).Name;

\_logger.LogInformation("Paaspop service Request: {Name} {@Request}", name, request);

return Task.CompletedTask;

}

}

### AutoMapper

Automapper zorgt ervoor dat objecten automatisch naar elkaar gemapt kunnen worden zonder dat hier altijd aparte methodes voor geschreven hoeven te worden. Ter voorbeeld, als eerste wordt er een profile aangemaakt waarin automatisch gemappt kan worden, maar ook zelf regels geschreven kunnen worden:

public class ArtistProfile : Profile

{

public ArtistProfile()

{

CreateMap<Artist, ArtistViewModel>();

}

}

Door middel van dependency injection kan deze mapper gebruikt worden zoals in een request waarin het resultaat van een Artist automatisch gemapt wordt naar een ArtistViewModel:

return Mapper.Map<ArtistViewModel>(result);

### FluentValidation

Door middel van fluent validation kunnen requests automatisch worden gevalideerd voordat ze uitgevoerd worden. Er kunnen zelf eisen worden opgesteld waar een request (zie GetArtistQuery object in 5.3.1) aan moet voldoen. Als een request niet aan de juiste eisen voldoet kan er een exceptie opgegooid worden. Een voorbeeld van een validator:

public class ArtistViewModelValidator : AbstractValidator<GetArtistQuery>

{

public ArtistViewModelValidator()

{

RuleFor(req => req.Id).NotEmpty().Length(24);

}

}

En om er voor te zorgen dat elke request gevalideerd wordt, is het volgende stukje code geïmplementeerd in de mediatr pipeline [[5]](#footnote-5):

public class RequestValidationBehavior<TRequest, TResponse> : IPipelineBehavior<TRequest, TResponse>

where TRequest : IRequest<TResponse>

{

private readonly IEnumerable<IValidator<TRequest>> \_validators;

private readonly ILogger \_logger;

public RequestValidationBehavior(IEnumerable<IValidator<TRequest>> validators, ILogger<TRequest> logger)

{

\_validators = validators;

\_logger = logger;

}

public Task<TResponse> Handle(TRequest request, CancellationToken cancellationToken,

RequestHandlerDelegate<TResponse> next)

{

var failures = \_validators

.Select(v => v.Validate(request))

.SelectMany(result => result.Errors)

.Where(f => f != null)

.ToList();

if (failures.Count != 0) LogErrors(failures);

return next();

}

public void LogErrors(List<ValidationFailure> failures)

{

foreach (var failure in failures)

{

\_logger.LogError(failure.ErrorMessage);

throw new CustomValidationException(failure);

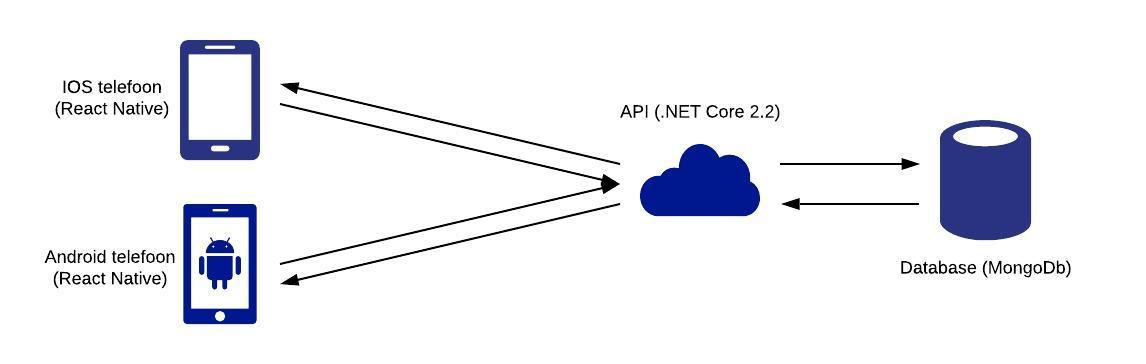
}

}

}

## Hoog level architectuur schets

Op een hoog niveau is er een architectuur schets gemaakt welke hier onder te zien is:



Via een client, in dit geval een IOS of Android telefoon, kan de gebruiker de react native paaspop app opstarten. Deze app staat in verbinding met de .NET core 2.2 API welke alle requests doorloopt en, waar kan, data verwerkt en terug / door stuurt.

1. (Martin, 2012) [↑](#footnote-ref-1)
2. (Battiston, 2016) [↑](#footnote-ref-2)
3. (Microsoft, 2018) [↑](#footnote-ref-3)
4. Deze requestlogger is niet zelfgeschreven maar is uit deels uit het volgende project overgenomen: https://github.com/JasonGT/NorthwindTraders [↑](#footnote-ref-4)
5. Deze validation pipeline is niet zelfgeschreven maar komt uit het volgende project: https://github.com/JasonGT/NorthwindTraders [↑](#footnote-ref-5)